2020

Activity recognition and animation of activities of daily living

Mohammed Shaiqur Rahman
Iowa State University

Activity recognition and animation of activities of daily living

by

Mohammed Shaiqur Rahman

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Carl K Chang, Major Professor
Simanta Mitra, Major Professor
Gurpur Prabhu

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2020

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

ACKNOWLEDGMENTS

I would like to thank my committee chair, Dr. Carl Chang and Dr. Simanta Mitra, and my committee member, Dr. Gurpur Prabhu, for their guidance and support throughout the course of this research. Also, special thanks to Dr. Carl Chang, for his guidance, patience and support throughout this research and the writing of this thesis.

In addition, I would also like to thank my friends, colleagues, the department faculty and staff for making my time at Iowa State University a wonderful experience.

ABSTRACT

Activities of Daily Living (ADL) can give us information about an individual's health, both physical and mental. They are captured using sensors and then processed and recognized into different activities. Activity recognition is the process of understanding a person's movement and actions. In this work, we develop a language in a simple grammar that describes the activity and uses it to recognize the activity. We call this language as Activities of Daily Living Description Language, or $A(DL)^2$ in short.

Even after an activity has been recognized, the data it represents is still digital data and it would take some expertise and time to understand it. To overcome this problem, a system that can visualize and animate individuals' activity in real time without violating any privacy issues, can be built. This will not only help in understanding the current state of individual but will also help those who are in charge of monitoring them remotely like nurses, doctors, family members, thereby rendering better care and support especially to the elderly people who are aging.

We propose a real time activity recognition and animation system that recognizes and animates the individual's activity. We experimented with one of the basic ADLs, walking, and found the result satisfactory. Individuals location is tracked using sensors and is sent to the recognition system which then decides the type of activity in real time by using the language to describe it, and then the data is sent to a visualization system which animates that activity. When fully developed, this system intends to serve the purpose of providing better health care and immediate support to the people in need.

CHAPTER 1. INTRODUCTION

By 2050, it is estimated that globally one in six people will be over age 65 [1] and in the United States, it is projected that the population over age 65 will increase to 71 million in 2030 and 98 million by 2060—when older adults will make up nearly 25% of the population [2]. Scientific breakthroughs and improved medical facilities have made it possible that the average life span of humans has increased. However, aging may have health complications like physical weakness, dementia to name a few, and therefore it demands better health care and monitoring especially to the elderly people who are aging in place. Monitoring their daily activities will not only help in providing immediate medical help but also over the time it may give some idea about their change in behavior that could point to an early sign of dementia. For example, consider fall among older adults, they are common, and it is the leading cause of fatal and non-fatal injuries [3]. Detecting the fall will reduce the time to get medical care, thereby reducing the fatality rate. In order to monitor these daily activities, motion sensors are used, however, to recognize these activities into different basic ADLs pose a challenge.

We propose a system that will recognize and animate the activity in real time. The system consists of a grammar that not only describes the activity but also recognizes it. After a particular activity has been identified, the resulting data is stored as a log for further analyses and the same data is communicated to Sketchup for animating that activity. Animating an activity using the exact model of the house with accurate measurements will not only give us a better way of understanding a person's behavior but it may also give us the exact location of a person in case of any accidents and the best way to reach that location.

The major feature of our system is to recognize the activity and then by some visualization tool animate that activity.

1.1 Activity Recognition

It is the process of understanding a person's movement and actions, and categorizing them into one of the basic ADLs like a person is walking, sleeping, eating, cooking etc. Activities of a person may give us some information about that person's behavior and intention. This is one of the reasons that activity recognition has become an important subject in the fields of security and healthcare domains. The information needed to recognize an activity are provided by the motions and sensors in the form of raw data. These data are then formatted and converted into a format suitable for the parser to parse them into different activities. The parser follows a grammar built from ANTLR® 4 tool.

1.2.1 ANTLR 4

It is an open-source tool used for building languages, tools and frameworks. It stands for ANother Tool for Language Recognition. It needs a grammar to generate the lexer, parsers and tree parsers. Syntax rules are taken care by the grammar and semantic checking is done manually by using the tree parsers.

1.2 Animation

For animating an activity, we are using Sketchup® pro tool. Sketchup is a licensed tool which are used to create 3-D models of buildings, interiors, furniture's, landscapes etc. with

accurate measurements. We have developed a plugin in Sketchup which is fed data in real time for animating the activity. The plugin is written in ruby language.

1.2.1 SketchUp

Before mentioning the internals of SketchUp that will be referred later in other chapters, a brief introduction of SketchUp is provided first.

There are various 3-D modelling tool available in the market and most of them are specific to different types of audiences. For example, graphic artists and animators prefer 3ds Max®, Blender® and Cinema 4D® while engineer prefers AutoCAD®. However, SketchUp is different in the sense it can be used by all type of audiences and it is easier to learn the basics and moreover any additional functionality can be added using plugins.

Moving to the internals, a single SketchUp file(*.skp) gives all the information about the current SketchUp model. Moreover, these model objects have classes that provide information about the current design. Figures 1.1 and 1.2 are taken from the book Automatic SketchUp [11].

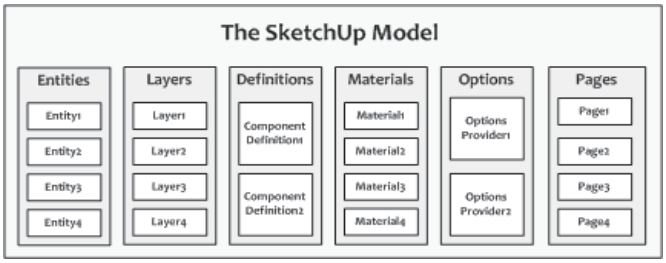

Figure 1.1 SketchUp model

Anything contained in a SketchUp model is entity. This is the base class for all entities and the hierarchy can be understood in the figure 1.2.

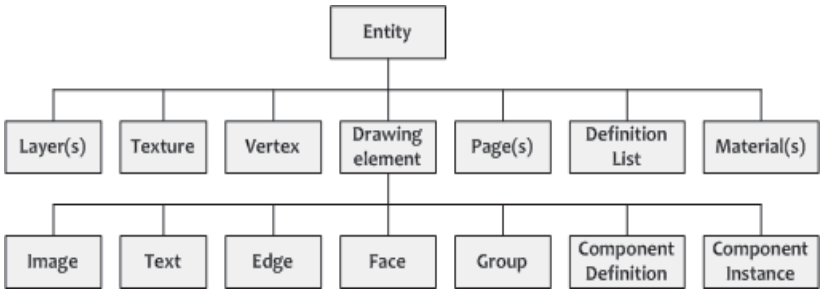

Figure 1.2 SketchUp Entity

1.2.2 Ruby

Ruby is interpreted and high-level programming language and supports both procedural and object-oriented programming paradigms. Although, SketchUp provides API's in both C and ruby, we selected ruby because the SketchUp tool itself is written in ruby.

Rest of the chapter are as follows. First Literature review of the key features in the field of activity recognition and animation where we explain a paper that describes a language for activity recognition. Then next chapter describes the system design that we have proposed, followed by implementation details and results. Finally, summary, followed by discussion about limitations, then future works before concluding.

CHAPTER 2. LITERATURE REVIEW

Activities of Daily Living (ADLs) refers to the activities that a person does independently to take care of oneself [4]. These activities may hint about a person's physical or mental health and inability to perform ADL means that the person needs assistance. Following are the basic ADLs types [4]:

- **Ambulating**: The ability to move/walk from one place to another.
- **Feeding**: The ability of a person to feed oneself.
- **Dressing**: The ability to put clothes on by selecting appropriate dress.
- **Personal hygiene**: includes bathing, grooming, dental, hair, nails etc.
- **Continence**: The ability to control bladder and bowel function.
- **Toileting**: The ability to get to and from the toilet, using it appropriately, and cleaning oneself

In addition to the above basic types, there are some Instrumental ADLs (IADLs) which requires more complex thinking. However, my research focus is on the basic types and particularly in the "Ambulating" type which will also be referred as walking or moving in the coming chapters.

There have been multiple works in the field of activity recognition and animation of the ADLs and to understand them in a better way, we divide them as:

1. Activity recognition
2. Animation

2.1 Activity recognition

This can further be divided into two categories.

1. Data analyses

2. Activity Description Language

2.1.1 Data Analyses

Multiple types of sensors and RFIDs have been used in smart home environment to detect user ADLs successfully. These sensors are costly, their initial setup is time consuming and after the setup, it needs continuous maintenance. In recent years, different approaches have been used to collect the user data, from using the smart phone as a single point for capturing ADLs [5], using Wi-Fi signals to detect falls and iBeacons for finding the local position of a person inside a room. This has greatly reduced cost of setup and has reduced energy consumptions. After the data is collected, it is analyzed to recognize a particular type of ADL.

2.1.2 Activity Description Language

These languages are specific in their usage and they can be used to describe an activity. Some of the existing languages like Scade, Lustre have been used in embedded systems as a critical control software. More recently a new language, ADeL [6] was proposed for activity recognition. This language is different from existing languages as it is easier to understand by non-computer scientist [6]. They have described each activity in the form of a program which is stored as a library (ADeL libraray). These libraries are called at runtime by their recognition system to recognize different activities. Fig 2.1[6] gives the general structure of their recognition system, following which are the brief description of their system architecture.

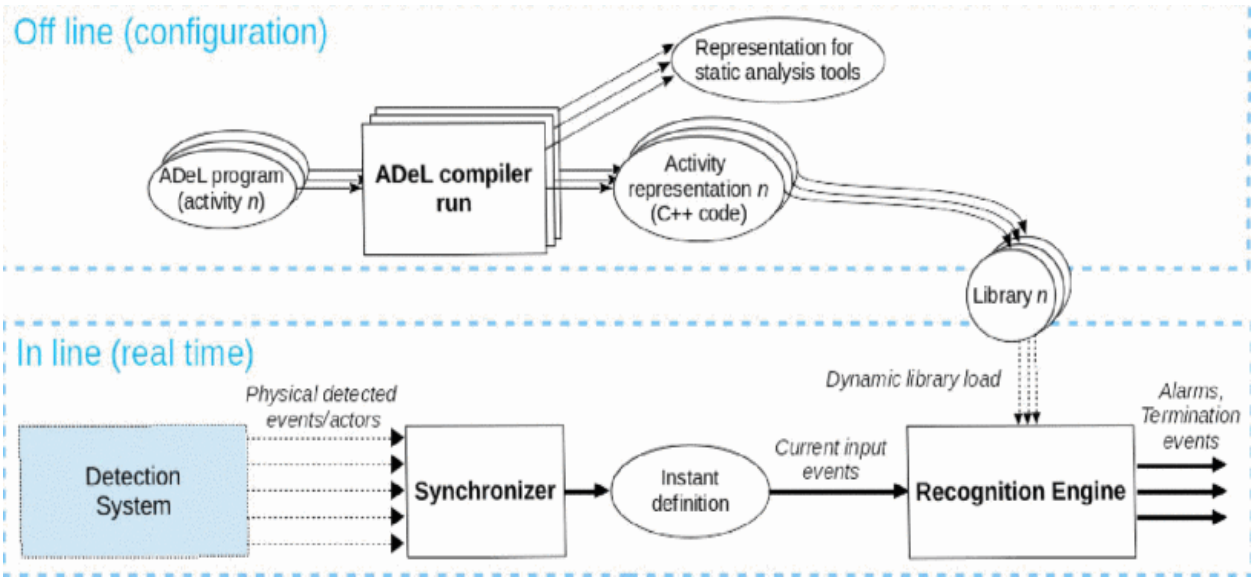

Figure 2.1 General structure of activity recognition system

2.1.2.1 Detection System

It continuously collects data from the sensors and identifies objects (actors) associated with the event and then forwards it in the format suitable for activity description.

2.1.2.2 Synchronizer

The main work of this module is to filter unwanted data and group simultaneous events into logical instants. These instants are defined as primitive events like a person is walking, sleeping, eating etc. These are then given as input to the recognition engine and their sequence defines the logical time.

2.1.2.3 Recognition Engine

It loads the ADeL library at run time and uses it to identify an activity. It also triggers output events like raising alarm or termination of activity.

2.2 Animation

Animating the activities by using the real time data is complex process as it involves both recognizing the activity and then animating it. Since the initial cost and the setup of smart home is large, there has been an alternate approach, where activities are simulated in virtual environment for generating the data sets for activity recognition research process. Persim 3D [7] is one these works which is a simulator capable of creating a virtual smart space and then generating data for research purposes. Yet another work involves collecting location data inside a room using android phones and WIFI and sending them to SketchUp for animating the movement [8].

2.3 Limitations

While both paper [6] and paper [7] are complete in their perspective, they both lack each other's capability. One lacks a language that would not only describe an activity but also recognize it at run time, while other lacks a virtual environment which would animate the activities in real time. In the next section we propose a system architecture that would combine both the ideas into single one, thereby providing much flexibility in the area of smart home where an activity is not only recognized but also visualized at run time.

CHAPTER 3. SYSTEM DESIGN

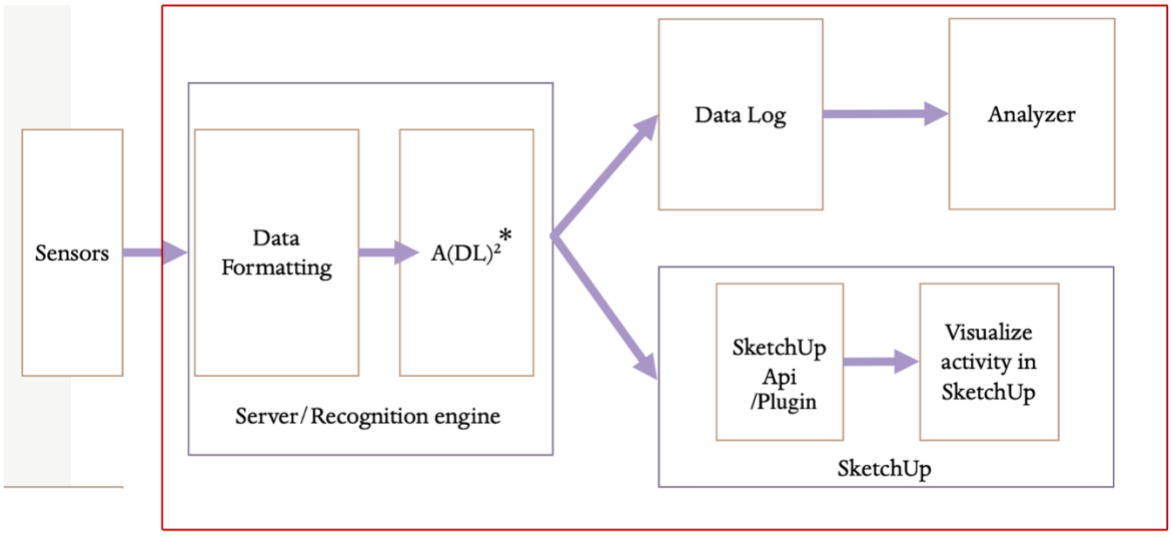

Figure 3.1 System design for the activity recognition and animation

Figure 3.1 gives the general structure of our whole recognition system. Activities of a person are constantly monitored using sensors and whenever there is some activity or event, the sensors record the data. These data are then forwarded to the server where our recognition engine resides, and this engine is responsible for recognizing the activity.

In the context of smart home lab, any activity could be assumed to have three basic units: actor, action and location.

actor: the one doing the action

action: determines the type of basic activity

location: any activity must have a location.

3.1 Sensors

By using Arduino or raspberry pi, data is collected from the sensors and sent to the server. For example, the location data collected from iBeacon in json format is as:


```
{
        "user-id": 154,
        "location": {
                        "x": 42,
                        "y": 66
                    },
  "numanchors": 4,
  "timestamp": "2013-01-04 09:09:39"
}
```


Explanation of the data fields:

user_id: - id of the person associated with the activity

location: - location of the person in “x” and “y” coordinate

numanchors: - number of iBeacons

timestamp: - the time at which event originated.

3.2 Recognition engine

It formats the data received from the sensors and recognizes the activity associated with that particular data. This module could be further divided into two categories.

3.2.1 Data Formatting

The json data is transformed into textual data which describes the basic activity associated with the data. For example, an activity could be described as walking if there are two location data with different coordinates and slightly different timestamps. Of course, the data should belong to the same user and this could be verified by checking the “user_id” tag in the data. It is then

converted into textual data as below and a resulting output from the program is also shown in Figure 3.2.

actor + action + location.

“user_id” + is + “walking”+ in+ “living room”

Figure 3.2 Showing data in textual format

The resulting text is then sent to the A(DL)^2.

3.2.2 A(DL)^2

This is the language(grammar) that recognizes an activity and in order to identify a given text into any activity, the text is first tagged using the Stanford NLP [9] library. For example, the text “user is walking” will be tagged as:

“154_CD is_VBZ walking_VBG”

Figure 3.3 Showing text after tagging

Figure 3.3 shows the output after it has been tagged by tagger using the program. Here “154” is the user_id and “CD” is the tagged value. Once the text has been tagged, it is parsed by

the grammar and the corresponding parse tree is generated, figure 3.4. Any error in the syntax is reported.

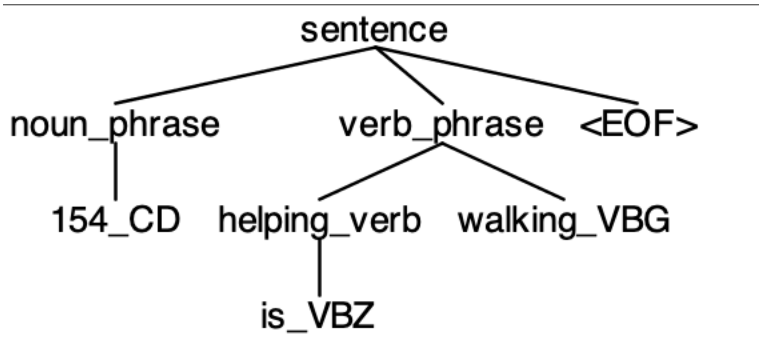

Figure 3.4 Showing the parse tree of the textual data

Currently the grammar is a general description of all the activities, and it is used to parse a given text in correct syntax order. Then to identify the type of activity, the tag value “VBG” is used. In order to make the grammar more specific to a particular activity, it could be restricted to have attributes because each activity has certain unique attributes to identify them.

This way of identifying the activity is not the best approach as it seems redundant work because we have already identified the activity while converting the data into text. However, it would be more useful if there are more complex activities. For example, there could be multiple activities happening simultaneously, then the grammar will have to identify them and distinguish them separately into different activity.

3.3 Data Log and Analyzer

Once an activity has been recognized, the corresponding data is stored in the database. These data could be further analyzed by some machine learning models to identify any change in the user's common behavior. This is very helpful in finding any early signs of dementia.

3.4 Sketchup

It continuously takes data from the server and animates the user activity. Animation is done by using avatar which mimics the user activity. SketchUp uses plugin to interact with the objects inside of it and also to communicate with the objects outside of it. These plugin acts as a tool which enhances the capability of SketchUp, and this could be designed according the requirement of user. For our research purpose, the plugin performs the animation by taking data from server and passing it the avatar inside the Sketchup.

CHAPTER 4. IMPLEMENTATION

My current research focus is only on one of the basic ADLs, walking and to implement that, the whole process has been divided into following phases:

1. Activity Recognition using Grammar
2. Database setup
3. Animation using SketchUp

4.1 Activity Recognition using Grammar

After receiving the raw data from the sensors, it needs to be formatted into a text, corresponding to the activity it represented, so that the text could be given as input to the grammar for activity recognition. The text corresponds to the basic activity like a person is walking/sleeping/cooking etc. The grammar takes these texts and recognizes them into different activities. These activities can be a complex activity like a person is cooking as well as doing dishes, eating and watching tv etc. or can be a simple activity like walking.

4.1.1 Grammar using ANTLR 4

The grammar used here is a simple grammar that parses any valid English sentences. It could be made more specific by providing attributes to it, so that it only parses ADLs. The grammar is shown in figure 4.1.

```
grammar Walk;

sentence: (noun_phrase verb_phrase)+ EOF
        ;
noun_phrase: noun_phrase preposition_phrase
           | ARTICLE NOUN
           | NOUN NOUN
           | NOUN
           ;

verb_phrase: verb_phrase preposition_phrase
           | helping_verb VERB
           | VERB noun_phrase
           | VERB
           ;

helping_verb: HLPVERB;

preposition_phrase: IN noun_phrase
                  ;

ARTICLE : ('a' | 'A' | 'An' | 'an' | 'The' | 'the')'_DT';
NOUN : [a-zA-Z]+'_PRP' | [a-zA-Z0-9]+'_CD' | [a-zA-Z]+'_NN' | [a-zA-Z]+'_NNP';
VERB : [a-z]+'_VBG';
HLPVERB : [a-z]+'_VBZ' | [a-z]+'_VBP' ;
IN : [a-z]+'_IN';
WS : [ \t\n]+ ->skip;
```

Figure 4.1 Showing the grammar

ANTLR 4 uses this grammar to create lexer and parser. The lexer tokenizes any input texts and the parser checks if the given text follows the rules of the grammar and any error is reported back.

ANTLR 4 requires a grammar structure to be followed so that it could read it properly. The first line starts with grammar followed by name of the grammar. In my case it is grammar Walk. Second line starts with the starting symbol of the grammar and ends with “EOF”. Here starting symbol is “sentence” and ANTLR 4 uses “EOF” to parse the whole input. If “EOF” is not present, then it may not parse the whole input to avoid any syntax error. Following are the grammar rules

(production rules) and lexer rules. Grammar rules are in lower-case letters while all the lexer rules are in upper-case letters.

The formal definition of the grammar could be given as a 4-tuple, walk= (V, Σ , R, S)

V =set of non-terminal symbols

{sentence, noun_phrase, verb_phrase, helping_verb, preposition_phrase}

Σ= set of terminal symbols

{ARTICLE, NOUN, VERB, IN, HLPVERB}

R = productions of the grammar

S = start symbol

{sentence}

However, in order to parse the text by the grammar, the text needs to be tagged first. This makes it easier for the ANTLR 4 tool to identify different words as different parts of speech. Thus, for tagging we have used Stanford POS tagger. The tagger is licensed under the GNU General Public License (v2 or later).

Although, using tagger did make the parsing smooth, it introduced a processing time of 0.5 to 1 sec depending upon the text it is tagging. This may not be ideal for a real time system.

Tag	Description	Example	Tag	Description	Example
CC	coordin. conjunction	*and, but, or*	SYM	symbol	+,%, &
CD	cardinal number	*one, two*	TO	"to"	*to*
DT	determiner	*a, the*	UH	interjection	*ah, oops*
EX	existential 'there'	*there*	VB	verb base form	*eat*
FW	foreign word	*mea culpa*	VBD	verb past tense	*ate*
IN	preposition/sub-conj	*of, in, by*	VBG	verb gerund	*eating*
JJ	adjective	*yellow*	VBN	verb past participle	*eaten*
JJR	adj., comparative	*bigger*	VBP	verb non-3sg pres	*eat*
JJS	adj., superlative	*wildest*	VBZ	verb 3sg pres	*eats*
LS	list item marker	*1, 2, One*	WDT	wh-determiner	*which, that*
MD	modal	*can, should*	WP	wh-pronoun	*what, who*
NN	noun, sing. or mass	*llama*	WP$	possessive wh-	*whose*
NNS	noun, plural	*llamas*	WRB	wh-adverb	*how, where*
NNP	proper noun, sing.	*IBM*	$	dollar sign	*$*
NNPS	proper noun, plural	*Carolinas*	#	pound sign	*#*
PDT	predeterminer	*all, both*	"	left quote	' or "
POS	possessive ending	*'s*	"	right quote	' or "
PRP	personal pronoun	*I, you, he*	(	left parenthesis	[, (, {, <
PRP$	possessive pronoun	*your, one's*	)	right parenthesis	],), }, >
RB	adverb	*quickly, never*	,	comma	,
RBR	adverb, comparative	*faster*	.	sentence-final punc	. ! ?
RBS	adverb, superlative	*fastest*	:	mid-sentence punc	: ; ... – -
RP	particle	*up, off*			

Figure 4.2 Penn TreeBank Tagset

4.1.2 Stanford POS Tagger

This reads text and assigns each word a part of speech such as nouns, verbs etc. It uses Penn TreeBank Tagset for tagging the words. For example, any verb will have an ending tag as "_VBG", any noun as "_NN" or "_PRP". Figure 4.2 shows the different tags that the tagger uses. This helped in recognizing text and thereby categorizing into activity.

4.2 Database setup

For the research purpose, MySQL database has been used. Currently, it has only two table, one for storing the last user details and other for storing the activity, user is associated with.

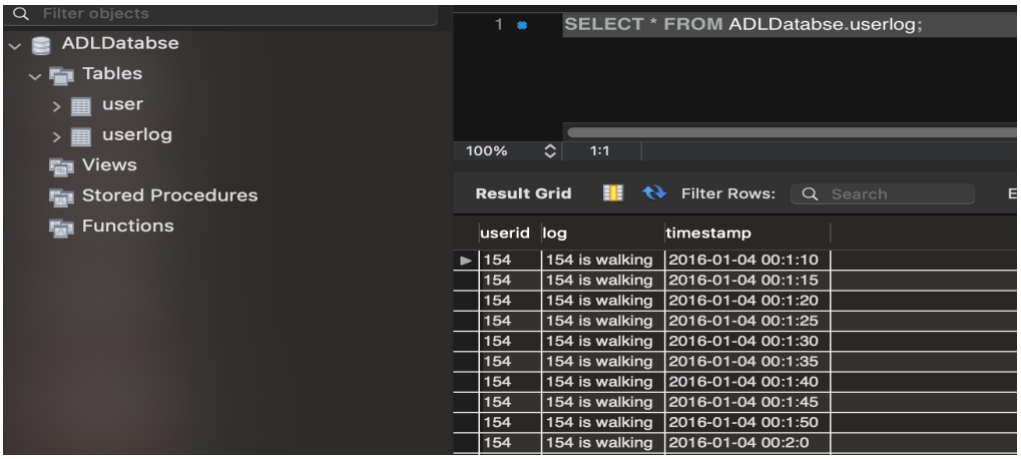

Figure 4.3 Showing the userlog table.

It stores user activity for the user and the tracking timestamp. These logs can be further analyzed by some machine learning modules to identify any uncommon behavior or change in the user's daily activities. This may be helpful in determining any early signs of dementia.

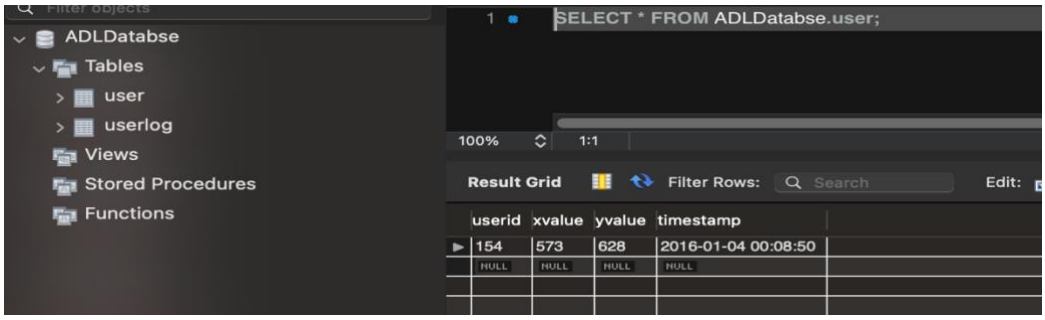

Figure 4.4 Showing the user table.

The user table stores the user location coordinates along with the timestamp at which it was tracked. This table is being queried by two modules: data formatting module which uses this table to check whether the location it has received is actually current location and the SketchUp module which constantly reads the table to get the current location coordinates of the user to animate the activity.

4.3 Animation using SketchUp

SketchUp is a 3-d modelling tool, built in ruby language. First the exact model of the place was built including the interior design. Then to access the model objects, we developed a plugin for the animation. We used ruby API [10] in the plugin since it is available from within the SketchUp.

4.3.1 Plugin

Plugins extends the functionality of SketchUp and allow building of custom tools according to the needs. We created a plugin to animate the user activity in real time by communicating over socket.

4.3.1.1 Socket connection

SketchUp acts as a client and makes request to the server using the sockets and in order to make the request continuously, we have used the SketchUp in-built method start_timer() which executes the same piece of code after a fixed amount of time and the process could be repeated several times. Figure 4.5 shows a code snippet.

```
stimer = UI.start_timer(1,true) {
  begin
    #puts "hello data is coming. "
    @socket.puts "Sketchup 154"
    line = @socket.gets
    if line.include? "location"
      puts line
      data = line.split(",")
      userid= data[1].delete(' ')
      x_pos = data[2].delete(' ')
      y_pos = data[3].delete(' ')
      timestmp = data[4].delete(' ')
      new_pos = Geom::Point3d.new(x_pos.to_f, y_pos.to_f, 0)
      @avatarModel.movetoPosition(new_pos)
```

Fig 4.5 Showing method start_timer

The method start_timer(seconds, repeat) has two parameters. First parameter is time in seconds before the code is executed again and second parameter is a Boolean value: true for repeating and false for not repeating and the default value is false.

This keeps the data coming and helps in animating the activity. For each new data that it receives, the avatar is moved to that location and then the active view is refreshed to reflect the changes.

SketchUp.active_model.active_view.refresh

4.3.1.2 Movement

Since SketchUp receives location coordinates, there is no issue in moving from one point to another as the distance between the two points are usually less than 20 inches. However, as SketchUp gets its data from the database which only holds the last user location information, the database may get updated by new contents before it has been read by SketchUp. In this case, if the

newer data points to a location which require turning the avatar because there is no straight path or there is some obstacle on the straight path, then the avatar needs to be rotated.

4.3.1.2.1 Rotation (θ)

If the new location to move is not in the current direction of movement then the avatar needs to be first rotated and the angle of rotation is calculated by using the below formula.

$$\theta = \sin^{-1}\left(\frac{\sqrt{(y2-y1)^2}}{\sqrt{(x2-x1)^2-(y2-y1)^2}}\right)$$

The angle (θ) is with respect to the x-axis when the location to move lies in the first quadrant (($x_2 > x_1$) and ($y_2 > y_1$)). Figure 4.6 shows the scenario to find the angle.

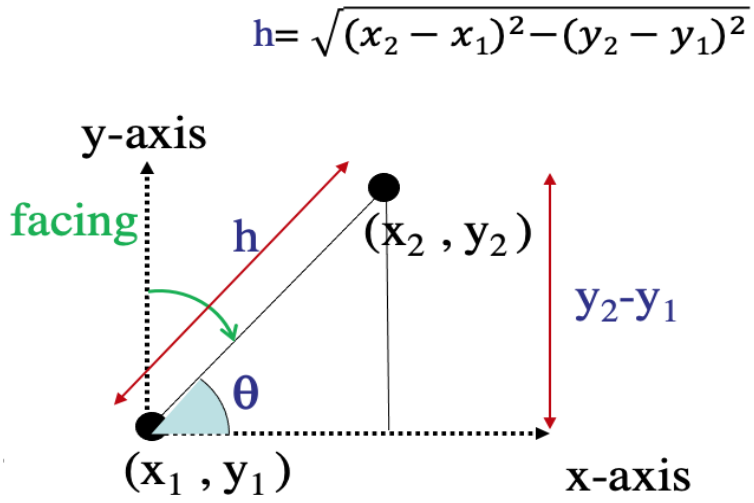

Figure 4.6 Angle calculation

Assume the avatar is at position (x_1, y_1) facing the y-direction and the position to move is (x_2, y_2). Since the new position is not in the same direction of facing (y-direction), the avatar needs

to be rotated to face the direction of new position. To do that we find angle θ with respect to x-axis and subtract it from 90. For rotation, SketchUp uses clockwise direction as positive.

However, movement after rotation becomes difficult. This is because every component that is created in SketchUp has its axis aligned with the global axis by default and when a component is rotated, its local axis also gets rotated as shown in the figure 4.7.

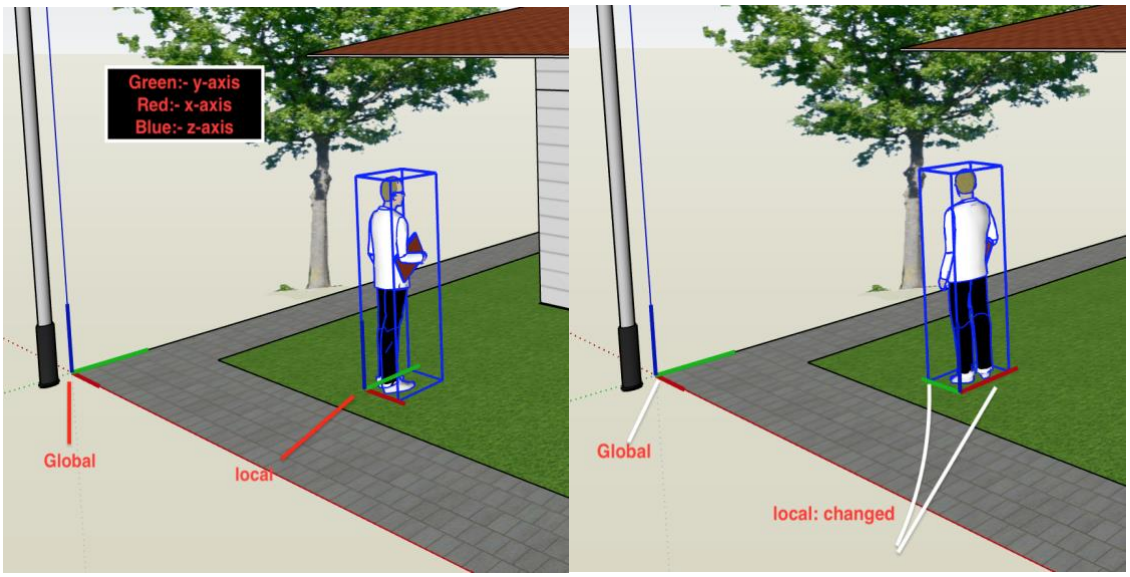

Figure 4.7 Left before rotation, right after rotation (clockwise by 90 degrees)

This rotation made the movement of component difficult because SketchUp gets confused between local and global axis. Two solutions were thought:

1. Change the local axis of the component to align with the global axis,
2. Change the global axis to align temporarily with the component axis.

But SketchUp does not allow changing the global axis of the model. While component axis can be changed from the SketchUp GUI, there is no way to change it from the ruby API. As none of the above solutions worked, an alternate approach was applied.

Explode the component into separate entities and recreate the component from those entities. This approach worked because creating any component, by default has its axis aligned with the global axis, thus allowing smooth movement.

4.3.1.2.2 Collision Detection

SketchUp does not provide any collision detection from ruby API, hence the workaround is to check whether a point lies inside or on the surface of a component and avoid that path if it does. It is easier to check if the point is inside the component, but to check if it is on the surface, all the faces in the entire model have to be considered for checking. Currently the model has 40476 entities, and each entity has one to six faces depending on whether it is 2-d or 3-d. Since the number of faces is very large and every few seconds, a new point needs to be checked, it would take some time to get the result and hence, it is not ideal for a real time animation.

Checking if a point lies within component is much faster as the number of components is much less than number of faces as components are collection of entities. Additionally, SketchUp provides a virtual cuboid called bounding box that surrounds a component, and an API to check if the point is inside the bounding box. Hence this approach has been implemented.

CHAPTER 5. RESULTS

To test the whole system requires real time data from the sensors but the sensors are not integrated yet, hence manual test data was used. Also, as the model is an exact replica of the house and SketchUp uses x, y and z coordinates to locate any object in the model, both the building and the model is assigned same origin. This is required because the data sent by sensors are in x, y, z coordinates. This helps in placing the objects accurately in the model. The focus of the testing is to be able to recognize and animate one basic ADLs - walking. The assumption is that the individual starts at some fixed location, say origin (0,0,0). The test data is generated by a test client which creates the location coordinates of the next position and sends it to the sever.

Walking is one of the most common ADLs and tracking this activity will help in locating the current position of the individual. This is important in the event if some accident happens like the individual has fallen. The animation will show that the person is not moving which may indicate something has happened.

We tested the animation of activity, walking and found the results as expected. We positioned the avatar at a starting location and then animated its movement based on the location coordinates sent by the test client. Although, the best way to see the results is to watch the animation, we have tired to give snapshots of animation here for the purpose of understanding. We have analyzed few scenarios.

5.1 Scenario 1

The person is walking from one point to another. This is the regular walk like a person is going to the kitchen or living room from the bedroom. The result of animating this scenario is shown in the figure 5.1. It shows logs from the server and a snapshot of the animation.

Figure 5.1 Showing a person is walking

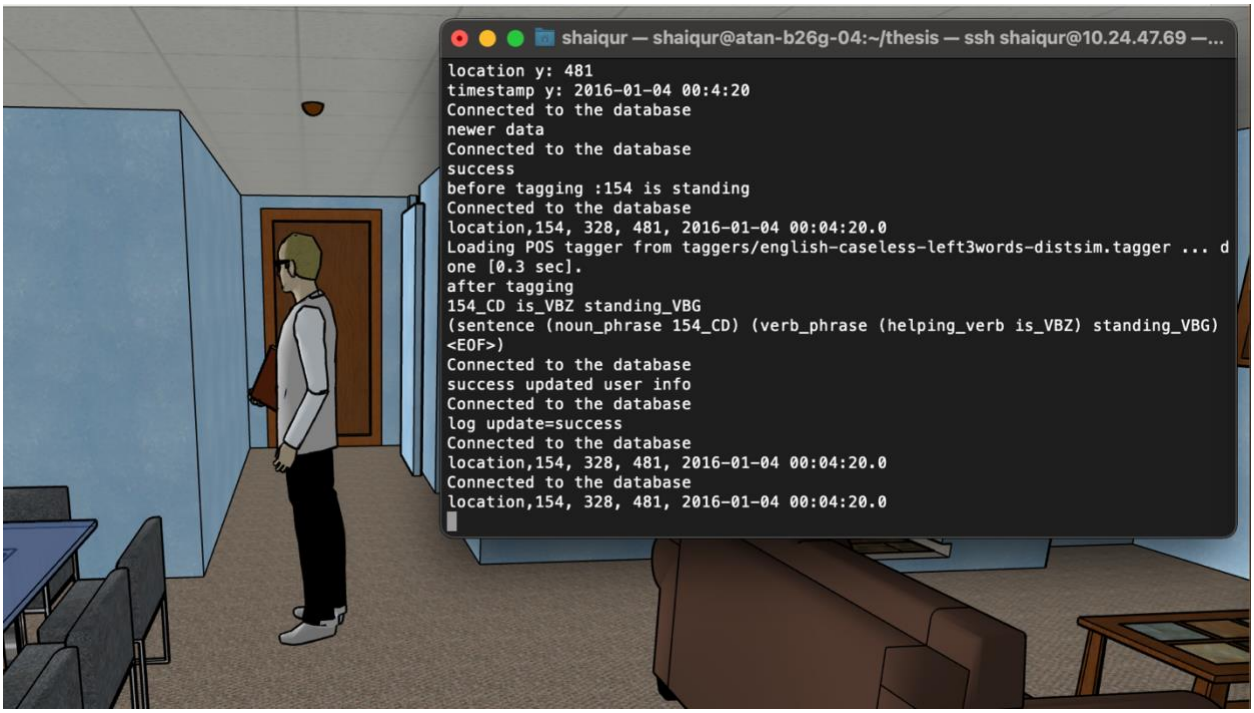

Figure 5.2 Showing a person is standing

5.2 Scenario 2

The person is standing as the time passes. This may mean the individual is just standing, but if the animation shows standing for a long time then it may indicate something has happened. Figure 5.2 shows the same. The server logs indicate the action.

Since, sensor will give the exact location of the individual, the data will never include any location inside any solid object. However, sometimes it will require for the avatar to turn left or right because the individual has changed its course of movement that requires rotation.

5.3 Scenario 3

The person walks and turns to some other direction. This is shown in figure 5.3 together with the server logs.

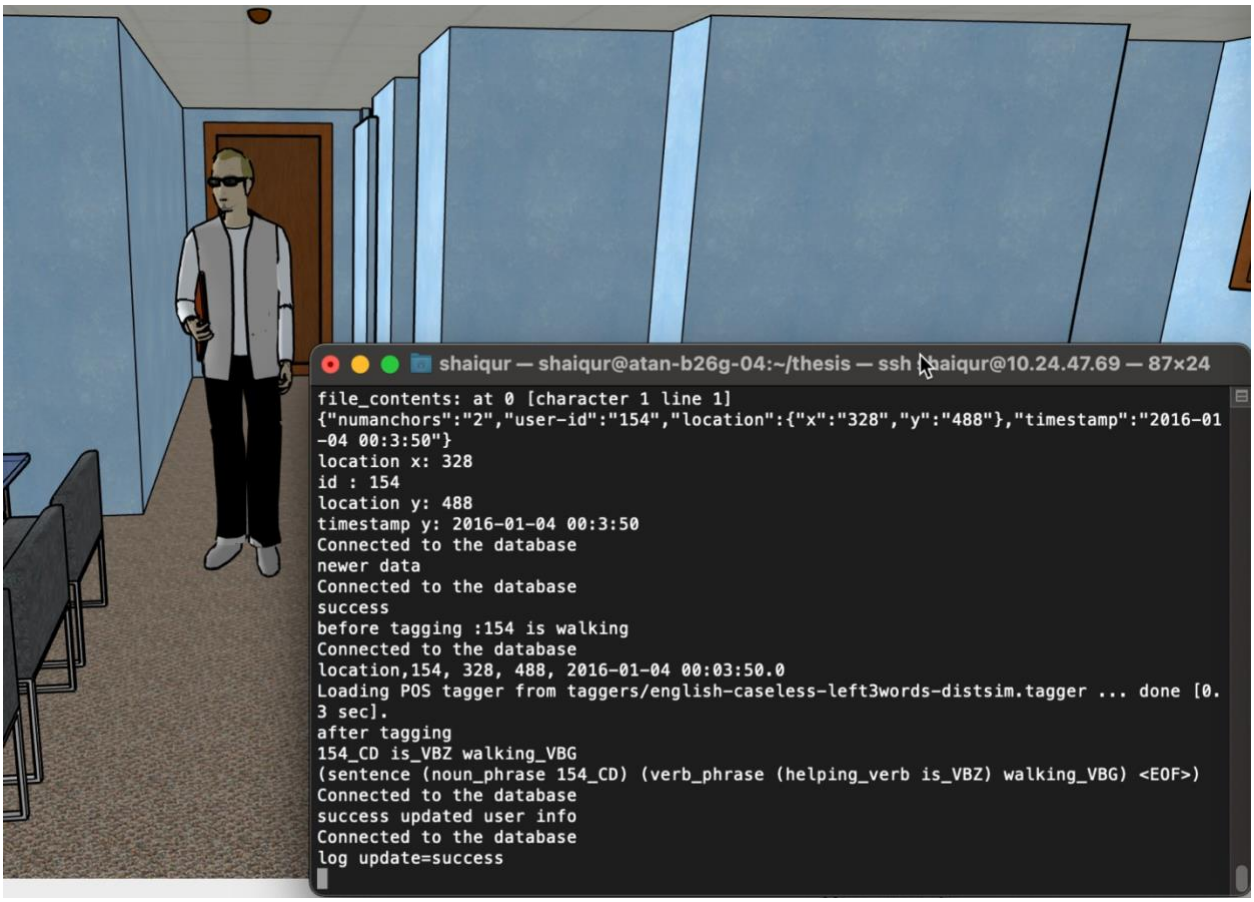

Figure 5.3 Showing walk after turning

CHAPTER 6. SUMMARY AND CONCLUSION

6.1 Summary

We designed a system which does the activity recognition and animation of the activity in real time. For recognizing the activity, we developed a language (grammar) which we called $A(DL)^2$. Though this language is not a complete language to describe an activity, it is a starting point which could lead to a better language in time. We studied the limitations of ADeL and Persim 3D and combined their ideas into one. And finally, we successfully animated one of the basic ADLs- walking, in the SketchUp with different scenarios.

6.2 Limitations

Although we have implemented the system design in the desired way, it suffers from various limitations. At first, there is no synchronizer that would synchronize the data it receives from the sensors in some logical time. Though, timestamp has been used to keep track of the sequence of events, we still need some way to store the data from various events before it is formatted as there could be events with same timestamps. This becomes more important when there are various activities happening simultaneously.

Currently, our implementation only stores the current status of the user location and if that gets updated before it has been read for animation, then the data is lost. It is not ideal to store every location details of a person every second into the database, hence a better approach could be applied to sort this issue. The use of POS tagger introduces a delay of 0.5-1 seconds and this is not preferred in a real time system. Our grammar is very generic as it describes all the activities, but

we need a specific grammar for a particular activity, so that it solely recognizes that activity. SketchUp too, has its own limitations, absence of collision detection features makes it difficult to move, issue of global and local axis etc.

Most of these limitations could be reduced, for example, by using some queueing technique like Rabbit MQ- synchronizer could be implemented, attribute grammar will probably make the grammar more specific which would mean POS tagger could be eliminated and finally instead of using SketchUp, some game engine like unity or blender could be used for animation.

6.3 Future Work

Future work would include making the system more robust by mitigating the limitations. We have tested only one ADL, but it could be upgraded to test other ADLs as well. Maintaining the security and privacy of the user data will also be a challenge since the data will be routed through cloud. A complete language could be developed that would ease the activity recognition process. Activity recognition also opens the scope of including user intentions. Hence, future activity could be predicted based on the current activity and the history of activities. This would be more useful in surveillance and security to detect any suspicious activity.

6.4 Conclusion

We designed a system that performs the activity recognition and animates the activity in real time. Our initial results are promising and although, this was only tested in one of the basic ADL (walking), we are confident that the same could be applied to other ADLs and even more complex ADLs.

REFERENCES

[1] UN, Department of Economic and Social Affairs, World Population Aging 2019

[2] CDC, Promoting Health for Older Adults.
https://www.cdc.gov/chronicdisease/resources/publications/factsheets/promoting-health-for-older-adults.htm

[3] CDC, Older Adult Falls Data.
https://www.cdc.gov/falls/data/index.html

[4] Peter F. Edemekong; Deb L. Bomgaars; Sukesh Sukumaran; Shoshana B. Levy, "Activities of Daily Living", Europe PMC, 2020

[5] Yunfei Feng, Carl K. Chang and Hua Ming, "Recognizing Activities of Daily Living to Improve Well-Being", IEEE IT Professional, (2017)

[6] Ines SARRAY, Annie RESSOUCHE, Sabine MOISAN, Jean-Paul RIGAULT and Daniel GAFFE, "An Activity Description Language for Activity Recognition", IEEE, International Conference on Internet of Things, Embedded Systems and Communications (IINTEC), 2017 (Gafsa, Tunisia).

[7] Jae Woong Lee, Seoungjae Cho, Sirui Liu, Kyungeun Cho, Member, IEEE, and Sumi Helal, Fellow, IEEE, "Persim 3D: Context-Driven Simulation and Modeling of Human Activities in Smart Spaces", IEEE Transactions on Automation Science and Engineering, 2015

[8] Real-Time Indoor Movement Animation System In 3D Environment, Weijia Zhao, ISU digital repository, 2019

[9] Kristina Toutanova and Christopher D. Manning. 2000. Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pp. 63-70.
https://nlp.stanford.edu/software/tagger.shtml

[10] "Sketchup Ruby API Documentation", Sketchup Developer.
https://ruby.sketchup.com/

[11] Scarpino, Matthew. Automatic SketchUp: Creating 3-D Models in Ruby, Eclipse Engineering LLC; 0 edition (March 8, 2010)